IST-2001-32133
GridLab - A Grid Application Toolkit and Testbed

Architecture Requirements

Author(s):	Jarek Nabrzyski, Andre Merzky, Technical Board
Document Filename:	GridLab-14-AR-0003-ArchReqs
Work package:	WP14 Project Management
Partner(s):	All
Lead Partner:	Poznan Supercomputing and Networking Center
Config ID:	GridLab-14-AR-0003-1.0
Document classification:	PUBLIC

Abstract: This document describes and evaluates the main requirements to the overall GridLab architecture. GridLab, as an application driven project, derives most of its design constraints from the application domain, but also from generally accepted Grid design objectives.

This paper is actually an excerpt from a submitted research paper [1]. The original paper contains additional information about how the architecture requirements and design constraints are applied to the overall GridLab architecture, and about the more detailed design of several GridLab components.

Contents

1 Introduction

Computational Grids are becoming increasingly common, promising ultimately to be ubiquitous and thereby change the way global resources are accessed and used. However, presently there is a dearth of real Grid users, in part because the whole concept is new, but also because few applications have been written that can exploit Grid resources. Although some application developers are interested in writing Grid-enabled applications, there are few user-level tools, high level tools for application developers are nonexistent, and well-understood Grid usage scenarios are rarely available.

In order to catalyze Grid usage, it is therefore imperative to attract real users into the Grid community (for example the Global Grid Forum (GGF)) and ultimately onto the Grid. The Applications Research Group (APPS-RG) of the GGF (and formerly of the European Grid Forum, EGrid) has been addressing questions about user requirements, problems, and usage scenarios for several years now. In 2000, the group established a pan-European testbed [2], based on the Globus Toolkit, for prototyping and experimenting with various application scenarios. These testbed experiences gave inspiration for an application oriented project, called *GridLab*, funded by the European Commission.

The primary aim of GridLab is to provide users and application developers with a simple and robust environment enabling them to produce applications that can exploit the full power and possibilities of the Grid. The GridLab project brings together computer scientists with computational scientists from various application areas to design and implement a *Grid Application Toolkit (GAT)*, together with a set of Grid services, in a production grid environment. The GAT will provide functionality through a carefully constructed set of generic high-level APIs, through which an application will be able to call the underlying Grid services. The project will demonstrate the benefits of the GAT by developing and implementing real application scenarios, illustrating compelling new uses of the Grid. We will make extensive use of specific application frameworks, namely Cactus [3, 4] and Triana [5], as powerful and broad reaching, real-world application examples for developing GridLab, but the GAT will be useful for many kinds of applications and users. Our aim is to make Grid computing accessible for the widest possible spectrum of applications and users.

The development of the GAT is accompanied by the establishment of a pan-European testbed (including real production machines for the respective application user communities) and by the

development of Grid services of varying complexity, tailored to the needs of our user community. These services are designed to complement and complete the existing Grid infrastructure, and to provide functionality needed by the GridLab applications in order to be usefully deployed in such environments.
In this paper, we first present our vision of Grid-empowered application scenarios. We then motivate and discuss the global architecture of the project. This is followed by a more detailed discussion of exemplary GridLab components: the GAT (the application interface to Grid environments), Triana (one of the GridLab applications), and the GridLab Scheduling Service (a GridLab service).

2 A Vision of Grid-Empowered Application Scenarios

The advocates of Grid computing promise a world where large, shared scientific research instruments, experimental data, numerical simulations, analysis tools, research and development platforms, as well as people, are closely coordinated and integrated in "virtual organizations". This integration will be fostered through web-based portals, woven together into modular wide-area distributed applications. One hypothetical scenario in astrophysics, described in the following, illustrates such an integration. Although sounding futuristic, many individual components have already been prototyped, and through the GridLab project we are striving to make such a scenario a common occurrence.
Gravitational wave detectors will rely on results from large-scale simulations for understanding and interpreting the enormous amounts of experimental data they collect. The Grid infrastructure is used both to share expensive and centralized resources among many scientists, as well as to integrate experimental data sources with the simulation codes necessary to analyze them. For example, the GEO600 detector in Hanover detects an event characteristic of a black hole or neutron star collision, supernova explosion, or some other cosmic event. Astronomers around the world are alerted and stand by, ready to turn their telescopes to view the event before it fades. However, the location of the event in the sky must first be found. This requires a time-critical data analysis with a number of templates created from full-scale simulations.
In a research institute in Berlin, an astrophysicist accesses the GEO600 portal and, using the performance tool, estimates the resources required for cross-correlating the raw data with the available templates. The brokering tool finds the fastest affordable machines around the world. Merely clicking to accept the portal's choice initiates a complex process by which executables and data files are automatically moved to these machines by the scheduling and data management tools. Then the analysis starts.
Twenty minutes later, on her way home, the astrophysicist's mobile phone receives an SMS message from the portal's notification unit, informing her that more templates are required and must be generated by a full-scale numerical simulation. She immediately contacts an international collaboration of colleagues who are experts in such simulations. Using a code composition tool in their simulation portal, her colleagues assemble a simulation code with appropriate physics modules suggested by the present analysis. The portal's performance prediction tool indicates that, due to memory constraints, the required simulation cannot be run on any single machine to which they have access. The brokering tool recommends that the simulation be run across two machines, one in the U.S. and the other in Germany, that are connected to form a large enough virtual supercomputer to accomplish the job within the required time limit. The simulation begins. After querying a Grid information server (GIS), the simulation autonomously decides to spawn off a number of time-critical template generating routines, and to run asynchronously on various other machines around the world.

An hour later, the network between the two machines degrades and the simulation again queries the GIS, this time deciding to migrate to a new machine in Japan while still maintaining connections to the various template generators at other sites. All the while, the international team of collaborators monitor the simulation's progress from their workstations or wireless devices from an airport (where several team members happen to be), visualizing the physics results as they are computed. The template data are assembled and sent to the GEO600 experimenter in Germany for analysis, which finally yields the likely source location for the gravitational wave signal. This triggers another Grid application which utilizes a different virtual organization and its infrastructure to direct the Hubble Space Telescope and various other available instruments toward this source location. The entire process, which could not be performed on any single machine or at any supercomputing site available today, takes only a few hours.

3 Requirements for a Grid Software Environment

The main goal of the GridLab Project is to provide a software environment for Grid-enabling scientific applications. It is our aim to provide an API through which applications access and use available resources. This API directly reflects application needs. Among the intended functionality is the exploration of available resources (CPU, storage, visualization, etc.); remote data access; application migration; etc. The API will be concentrated in the *Grid Application Toolkit* (GAT). The functionality behind the API will be provided by interchangeable capability providers, which may be GridLab services or third-party services.
In this section, we will briefly define important types of capability providers. We then summarize application requirements and general constraints on a software architecture for the Grid Application Toolkit, and its capability providers.

3.1 Terminology

This subsection reviews the terminology used throughout this paper. During the development of the GridLab architecture, we realized that the standard set of terms used in Grid Environments was insufficient to represent the whole range of our component types. Thus we tried to refine these definitions, hopefully without introducing incompatibilities to the original terms:

Capability Provider: A capability provider is an entity providing a specific capability. It is defined in terms of an interface used to invoke a capability, and the behavior expected in response to that invocation (i.e., capability provider = interface + behavior).

Service: "A service is a **network-enabled entity** that provides a specific capability. [...] A service is defined in terms of the protocol one uses to interact with it and the behavior expected in response to various protocol message exchanges (i.e., service = protocol + behavior)." [6]

Web Service: "The term 'web services' describes an important emerging distributed computing paradigm [with] focus on simple, Internet-based standards (e.g., eXtensible Markup Language: XML [...]) to address heterogeneous distributed computing. Web services define a technique for describing software components to be accessed; methods for accessing these components; and discovery methods that enable the identification of relevant service providers." [7]
service providers." [7]

Grid Service: "A Grid service is a web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgrade-ability." [7] Grid services are defined by the emerging OGSA standard.

GridLab Service: A service provided by the GridLab project. Where possible these will be Grid (OGSA) services.

Third-party Service: A service provided outside the scope of GridLab, either from underlying Grid middleware, from legacy software, or from concurrent developments outside of the GridLab project.

Adaptor: The adaptor pattern provides programmers with interfaces. Adaptor components implementing an interface abstract other components, which can have a wide variety of interfaces. Adaptors are vital inside the GAT to allow interfacing applications to heterogeneous capability providers.

3.2 Requirements

The ultimate goal of the GridLab project is to provide application programmers and users with an environment that enables scenarios such as the one from Section 2. From such scenarios, the main application requirements to the GridLab architecture can be drawn. A number of additional user requirements have been specified by the GridLab application groups. They mainly deal with the usability of the whole system and, in general terms, express their wish for some degree of involvement in the internals of the environment. It may not hold for all groups, but there are for sure users, but more importantly application programmers, who would not accept working with a complex system if that system presents itself as a black box *only*.
Other, more technical requirements are also to be applied to our project, originating from the type of environments our system wants to enable, and from general arguments about administration, security and such. This subsection lists these requirements, which are the base for the definition of the GridLab architecture, covered in Section 4.

(i) abstraction of the environment :
By definition, the ultimate requirement to the GAT is to provide an abstraction of the underlying Grid infrastructure, its services, and its communication layers.

(ii) adaptivity to the environment :
One of the most important requirements of the GridLab user community is that applications utilizing the GAT should be able to run in a wide variety of real world environments. These include the Grid environments of both present and future, disconnected environments (e.g. laptops, developer machines), and firewalled resources.

(iii) interchangeability of capability providers :
Application executions, whether inside a Grid installation or upon isolated, disconnected resources, should become alternate cases of a unified application, supporting execution upon whichever resources are available There should not be a separation between "normal" and a "Grid-enabled" versions of the application code. This requirement demands a single GAT-API with which applications can be developed. The capability providers relevant to the current configuration can then be instantiated at runtime.

(iv) complete control on all levels :
Our users, and even more our application developers, strongly request the ability to control the utilized environment to as much an extent as possible. This does not mean that the application programmer herself wants to take care of every detail of the environment, but rather enables the programmer to detect errors, to prototype, and to split complex operations into smaller, simpler pieces if necessary and so on.

(v) smart adaptivity on all levels :
Grid environments are ever changing, and very heterogeneous, which is a constant challenge to Grid middleware. Our solutions have to be able to adapt to such environments.

(vi) robustness and fail safety, error tracing facilities :
Complex systems as computational Grids and distributed software systems inherit multiple points of failure. The middleware has to be able to recover from failures gracefully, and to report failures. Application programmers as well as administrators need to be able to verify the systems functionality on all levels.

(vii) independence from communication/architecture models :
With the advent of the Open Grid Service Architecture (OGSA) [7], GridLab's architecture and services will revolve around OGSA's notion of "Grid services." For the time being, this seems the most widely accepted architecture and communication model for the Grid community. However, in order to enable support both for legacy services and future development, the GridLab architecture itself should be independent from any specific architecture/communication model (e.g. OGSA, Legion, CORBA, RMI, RPC etc.).

(viii) cleanly layered architecture :
To achieve a complete abstraction of the underlying (Grid) environment, a cleanly layered architecture seems appropriate.

(ix) security mechanisms on a global level :
In order to get accepted in a wider community and to reach production state, the general GridLab architecture must enforce a flexible but tight global security model.

(x) third-party services are to be easily incorporated :
The GridLab project will by no means be able to provide all types of services to its user community – but other groups and projects do and will continue to provide various Grid components useful to the users. Also, we hope that the lifetime of the GridLab architecture by far exceeds that of the project itself. For these reasons, the easy utilization of third party services is crucial to the success of our approach.

4 The GridLab Architecture

The GridLab project aims to bridge the gap between applications and Grid middleware by providing a software layer in between. We will now describe the architecture of this software layer, consisting of the Grid Application Toolkit (GAT) and the GridLab services.

4.1 Introduction - Mandates versus Policies

In some respects, the above requirements and constraints partially contradict each other. In particular, the mandate to incorporate security mechanisms for all components at a global level

(ix) is incompatible with support for third-party services (iii, x) and disconnected environments. To elaborate, it is difficult to enforce security policies for third-party services; furthermore, Grid security fundamentally depends on network access. For third-party services, this contradiction might be resolved by requiring accesses to any third-party service to be performed via a (secured) GridLab service. But this is not possible if the application is running in a minimalistic environment (ii). It is also impossible if the application deliberately chooses to contact a suspicious (e.g., legacy) component on its own behalf, as applications need full control on all levels (iv).

In fact, contradictions of this type are very common in Grid environments. Middleware developers want to have control over all aspects of the environment. This is especially true for security issues, but also for scheduling, network interfaces, communications, and so on. In turn, the end user desires the ability to run an application in any environment, benefiting from a Grid infrastructure if available, but also ignoring Grid resources if appropriate. In particular, our experience shows that the users must be able to deliberately ignore aspects of the environment without loosing the benefits of the environment at large. For many user communities, an "all–or–nothing" bargain is not acceptable.

In the GridLab project, we explicitly distinguish between abstract architecture concepts and specific realizations of these concepts. Our *"General Architecture"* reflects this effort. By design, our architecture allows the incorporation of global security (ix) and the encapsulation of the communication models (vii) by providing a layered (viii) framework for all its components. On the other hand, such specifications are based only on policies, not mandates. Consequently, developers and users are encouraged to utilize these features, but are also free to ignore these and to incorporate components incompatible with these policies where necessary (ii, iii, iv, vii).

As a result of these considerations, the general GridLab architecture consists of the overall design (Figure 1) and a set of implementation policies. This combination of design and policies is the result of intense and controversial discussions. This approach is considered a main characteristic of GridLab, attributable to the strong involvement of real application groups.

4.2 GL Architecture – Global Design

As discussed above, the overall architecture diagram, presented in Figure 1, defines a cleanly layered environment providing an abstract view for the applications. The applications, located on the highest level of the user space, can access *all* capability providers they need via the Grid Application Toolkit (GAT) API. The GAT also resides in user space, providing interfaces to the capability providers in the capability space.

A detailed discussion of how the GAT is intended to provide a set of well defined capabilities by accessing the various capability providers is presented in the original paper. For the present discussion, only some features of the GAT are important: the ability to access various kinds of capability providers via adaptors specific to their communication mechanism, and the semantics of invoking specific capability providers.

As discussed above, the GridLab project distinguishes between GridLab services and third-party services (e.g. low-level Grid services like GIS or GRAM, system services, libraries). Following the requirement (iv), the GAT is able and allowed to access all types of capability providers, on all levels.

4.3 GL Architecture – Implementation Policies

In addition to the presented diagram, a set of policies ensure that the outcome of the GridLab project itself consists of compatible, uniform, and consistent components. Deviations from these

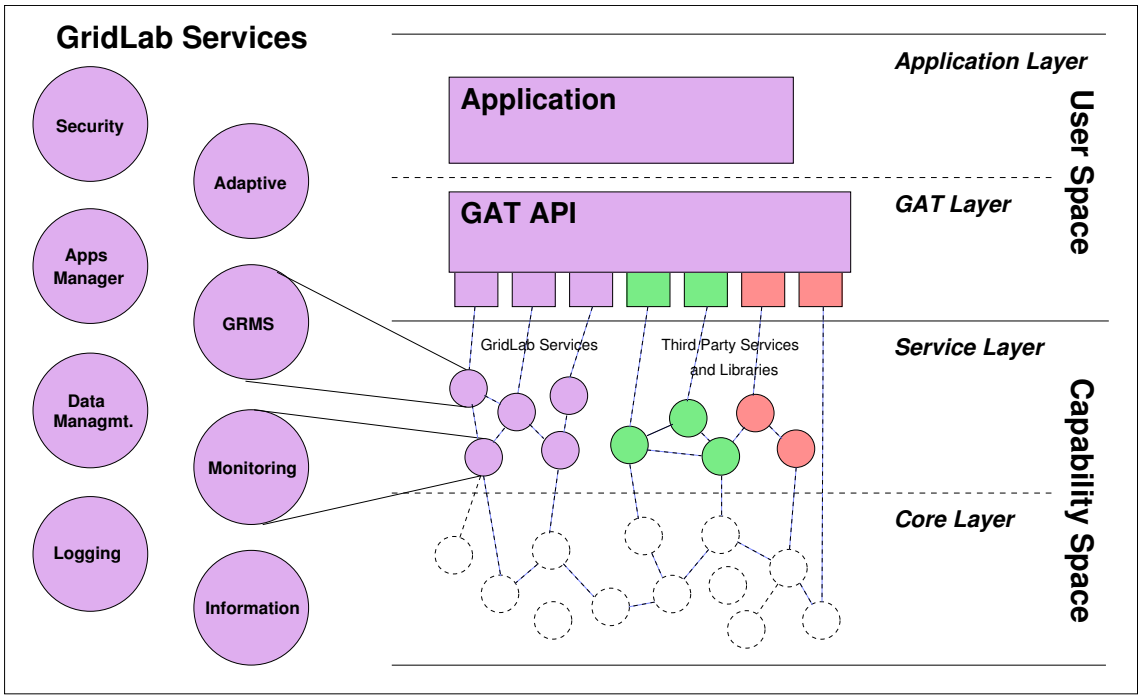

Figure 1: **The General GridLab Architecture:** The GAT will be designed to interact with all types of capability providers, via various communication channels. Security and uniformity for both types of components are recommended, not mandated (see text).

policies within the GridLab project require explicit justification. The implementation policies cover the following areas:

(1) Communication Model
All capability providers designed and implemented in the scope and course of the GridLab project are, by default, required to be *Grid Services*. However, until OGSA implementations become widely available (in particular for C++), GridLab services are being implemented as *Web Services* and will be converted to Grid services as soon as possible.

(2) Capability Access
The GAT is required to access capabilities via GridLab services, wherever possible and sensible. GridLab services are to be tailored to the needs of the GAT API. GridLab services will utilize lower-level services to implement their capabilities (like smart adaptivity, control, and fail safety).

(3) Security
All components designed and implemented in the scope and course of the GridLab project honor the GridLab Security Infrastructure.

These policies support the development of a clearly layered, consistent design of the GridLab software. Also, the combination of a general architecuture with strict policies preserves the simplicity and flexibility of the framework itself, which is necessary for the overall success of both the GridLab approach and the GAT beyond the lifetime of the project itself.
The original research paper contains a number of additiona sections describing various components of the presented architecture in more detail: the GAT itself, one of the major Gridlab

applications, Triana, and a sample GridLab service, the scheduling service.

5 Conclusions

In this paper, we have presented the overall architecture of the GridLab project, which aims to provide application-oriented Grid services for users and developers alike, covering the whole range of Grid capabilities as required by applications, such as resource brokering, monitoring, data management etc. These services will abstract lower-level Grid functionality and will hence ease the development and deployment of Grid-aware applications. This consistent service infrastructure will be the first major deliverable of the GridLab project.
These services will be made accessible to any applications running on the Grid through a Grid Application Toolkit (GAT), the second main project deliverable. The GAT will abstract those services needed by the Grid applications. In this way, applications can utilize service discovery at runtime, making use of whatever services are available, including different implementations of the same service. This will enable users and application developers to easily develop and run powerful applications on the Grid, without having to know in advance what the runtime environment will provide. Such applications should then both run on a single laptop or on an intercontinental Grid, taking advantage of whatever services are actually available (or unavailable). In particular, the GAT will not depend on the existence of any specific GridLab services.
Although we collaborate with the developers of powerful application frameworks such as Cactus and Triana for the development of GridLab, the project is designed to enable *any* application not only to run on the Grid (or without a Grid), but to endow it with new capabilities uniquely available on a Grid, such as those described above in our usage scenario. With such an architecture, we expect many new and powerful applications to be developed to exploit the Grids of today and tomorrow alike.

References

[1] G. Allen, K. Davis, K. N. Dolkas, N. D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor, "Enabling Applications on the Grid – A GridLab Overview," *International Journal of High Performance Computing Applications (IJHPCA)*, 2003. submitted on invitation.

[2] G. Allen, T. Dramlitsch, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, T. Kielmann, K. Verstoep, Z. Balaton, P. Kacsuk, F. Szalai, J. Gehring, A. Keller, A. Streit, L. Matyska, M. Ruda, A. Krenek, H. Frese, H. Knipp, A. Merzky, A. Reinefeld, F. Schintke, B. Ludwiczak, J. Nabrzyski, J. Pukacki, H.-P. Kersken, and M. Russell, "Early experiences with the EGrid testbed," in *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, (Brisbane, Australia), pp. 130–137, May 2001.

[3] Cactus Computational Toolkit home page: `http://www.cactuscode.org`.

[4] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, "The Cactus Framework and Toolkit: Design and Applications," in *Vector and Parallel Processing – VECPAR'2002, 5th International Conference*, (Berlin, Germany), Lecture Notes in Computer Science, Springer, 2003.

[5] Triana home page: `http://www.triana.co.uk`.

[6] S. Tuecke, I. Foster, and C. Kesselman, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, no. 3, 2001.

[7] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration." Draft Document, `http://www.globus.org/research/papers/ogsa.pdf`, June 2002.